AFRL-IF-RS-TR-2006-298
In-House Interim Technical Report
October 2006

GRID COMPUTING FOR HIGH PERFORMANCE COMPUTING (HPC) DATA CENTERS

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

NOTICE AND SIGNATURE PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)* OCT 2006	**2. REPORT TYPE** In-House Interim	**3. DATES COVERED** *(From - To)* Feb 04 – Jun 06

4. TITLE AND SUBTITLE GRID COMPUTING FOR HIGH PERFORMANCE COMPUTING (HPC) DATA CENTERS	**5a. CONTRACT NUMBER** In-House
	5b. GRANT NUMBER N/A
	5c. PROGRAM ELEMENT NUMBER 62702F
6. AUTHOR(S) Virginia W. Ross, Zenon Pryk, Walter Koziarz and Scott Spetka	**5d. PROJECT NUMBER** 459T
	5e. TASK NUMBER GR
	5f. WORK UNIT NUMBER ID
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) AFRL/IFTC, 525 Brooks Road, Rome NY 13441-4505	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFRL/IFTC, 525 Brooks Road, Rome NY 13441-4505	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSORING/MONITORING AGENCY REPORT NUMBER AFRL-IF-RS-TR-2006-298

12. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA# 06-688

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This research project developed an HPC grid infrastructure to operate as an interactive R&D tool. Previous HPC grid architectures were designed for batch applications, where the user lacks direct control over application execution. While this may be acceptable for running very large computationally intense atch jobs on large mainframes, it is not acceptable for use with jobs that require a near real-time response and an interactive environment. To meet this need, IFTC developed an environment that stresses 'near real-time' user interaction with the application. This involved evaluating the various developing protocols for interactive grid computing, using the Globus Toolkit, and then selecting the one with the most growth potential. The grid architecture evaluated by assembling and demonstrating an in-house interactive demonstration grid using in-house cluster assets and existing code, to verify proper operation on a small scale. This project provided a framework for longer term efforts to investigate and improve interactive scalability and performance, taking into account the needs of JBI, HPC, logistics, etc.

15. SUBJECT TERMS
Grid computing, interactive computing, high performance computing

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Virginia Ross
a. REPORT U	**b. ABSTRACT** U	**c. THIS PAGE** U	UL	22	**19b. TELEPHONE NUMBER (*Include area code)***

ABSTRACT:

This research project investigated techniques to develop a High Performance Computing (HPC) grid infrastructure to operate as an interactive research and development tool. Current HPC grid architectures are designed for batch applications, where users submit their job requests, and then wait for notification of job completion. In the batch environment, the user lacks direct control over the execution of their application. While this may be acceptable for running certain very large computationally intense batch jobs on large mainframe HPCs, it is not acceptable for use with jobs that require a near real-time response and an interactive environment. To meet this need for accessing and processing data interactively in near real-time, the Air Force Research Laboratory/ Information Directorate developed an environment that stresses 'near real-time' user interaction with the application. This involved evaluating the various developing protocols for interactive grid computing, using the Globus Toolkit, and then selecting the one with the most growth potential. The grid architecture was evaluated by assembling and demonstrating an in-house interactive demonstration grid using in-house cluster assets and existing code, to verify proper operation on a small scale. This project provided a framework for longer term efforts to investigate and improve interactive scalability and performance, taking into account the needs of the Joint Battlespace Infosphere, high performance computing, and logistics.

TABLE OF CONTENTS **Page**

LIST OF FIGURES **Page**

1. CONCEPT:

This research project investigated techniques to develop a High Performance Computing (HPC) grid infrastructure to operate as an interactive research and development tool. Current HPC grid architectures are designed for batch applications, where users submit their job requests, and then wait for notification of job completion. In the batch environment, the user lacks direct control over the execution of their application. While this may be acceptable for running certain very large computationally intense batch jobs on large mainframe HPCs, it is not acceptable for use with jobs that require a near real-time response and an interactive environment. With the current proliferation of HPC clusters and the existence of the internet, which allows for remote access to compute resources, the demand for rapid response times is escalating and achievable. To meet this need for accessing and processing data interactively in near real-time, AFRL/Information Directorate (AFRL/IF) developed an environment that stresses 'near real-time' user interaction with the application.

The approach for developing such an HPC grid capability involved evaluating the various developing protocols for interactive grid computing, and then selecting the one with the most growth potential. The Globus Toolkit was selected for the grid development environment. This toolkit is rapidly becoming the de-facto standard for grid research. The grid architecture was evaluated by assembling and demonstrating an in-house interactive demonstration grid using in-house cluster assets and existing code, to verify proper operation on a small scale.

Potential users of this product include developers of experimental systems as well as users who want to showcase their codes by making them available through the Hyperspectral Image Exploitation (HIE) Framework. [1-7, 9-13]. The primary emphasis was on creating a working interactive grid for use with the HIE framework. This project provided a framework for longer term efforts to investigate and improve interactive scalability and performance, taking into account the needs of the Joint Battlespace Infosphere (JBI), HPC, logistics, etc. It is envisioned that user interaction will be web based, to assure extensive portability.

To validate successful operation of the interactive grid implementation, the Hyperspectral Image Exploitation (HIE) framework software, was used [8]. Truth data already exists for this code, thus simplifying the final data analysis. This architecture also has the potential to be applied to streamline the management of Air Force logistics, operations where legacy software is widely used.

The major payoffs potentially include: Setting the groundwork for grid-based application of the HIE framework, improved real-time feedback for decision analysis tools, reduced latency for publication of subscription based data queries, and decreased turn-around time for decision support tools, wargaming, and modeling and simulation tools. An interactive grid gives the applications a more responsive underlying architecture to leverage future grid connected hardware.

The limitations for Grid computing over HPC Centers mostly arise from security concerns and the requirement to access heterogeneous environments. Even when systems are identical, the grid mechanisms that deal with security were difficult to implement and couldn't be adequately addressed within the scope of this project. Also, significant additional problems arise when hardware and software heterogeneity are considered. Other limitations are related to the difficulty in assigning processing and establishing reservations across groups of systems, for example at distinct HPC centers. Considerable additional work is necessary to consider these issues. This project explored the basic Grid capabilities, leaving these additional issues for future consideration.

2. APPROACH:

Three in-house workstations were configured for software development. Two were Linux based and one ran Microsoft Windows. Of the two Linux workstations, one was configured as a "grid" server. Communication between the server and the development workstation was established over a secure channel using Virtual Network Computing (VNC) based on Tight VNC and using a Secure Shell (ssh) connection from a Linux workstation running Red Hat Linux. Tight VNC supplied the ssh and graphical (X-based) connection to the server. There was an issue with resource intensive connections using Tight VNC.

To address this issue, the hardware was upgraded from the original 'grid' server, named "mintho" with a 1.2 GHz CPU and 512 MBs of random access memory (RAM) to a new server named "styx" with 2 central processing units (CPUs) operating at 2+ GHz and 4 GBs of RAM. It was hosting the installation of the VNC Server software and the Globus Toolkit (version 3.2). The various required ancillary tools were installed. [14] Graphical secure communication (using VNC) between mintho and a Windows 2000 desktop system and a Linux desktop system was demonstrated. The team evaluated the intricacies of Globus installation and determined an effective way to install Globus version 3.2 on the server. This provided access to all the necessary and sufficient tools to submit and interact with Grid (-like) job submissions. The next step was integration of a web-browser interface to Globus, providing the ability to launch, interact-with, terminate, etc. grid applications in semi-real-time. The Air Force Research Laboratory, Information Directorate (AFRL/IF), successfully integrated the Globus Grid toolkit with the HIE framework. One of the framework objects was rewritten as a grid service and presented at the 2005 Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA) Conference [1]. Additional work explored using Grid technology and information management for command and control [14].

The architecture diagram below, Figure 1, depicts the experimental setup used for the project. In the diagram, the virtual network computer (VNC) connection provides access from a Linux computer (astro) or Windows 2000 computer to the Grid Services Container. A “secure shell” (ssh) connection was also used to tunnel messages between Linux and the Globus “container” (server) system. The diagram shows a “counter” service operating within the Globus Grid Services Container. It shows that “stubs”, compiled into client software, provide access to the services that are active in the Grid Services Container. Services are accessed via Skeleton interfaces that are presented by the Grid Container as Grid services. The stub/skeleton basis for the architecture relieves developers of the requirement to implement detailed network interfaces, providing a significant opportunity for developers to implement efficient interfaces easily. The stub and skeleton are automatically generated from a Web Services Description Language (WSDL) interface description.

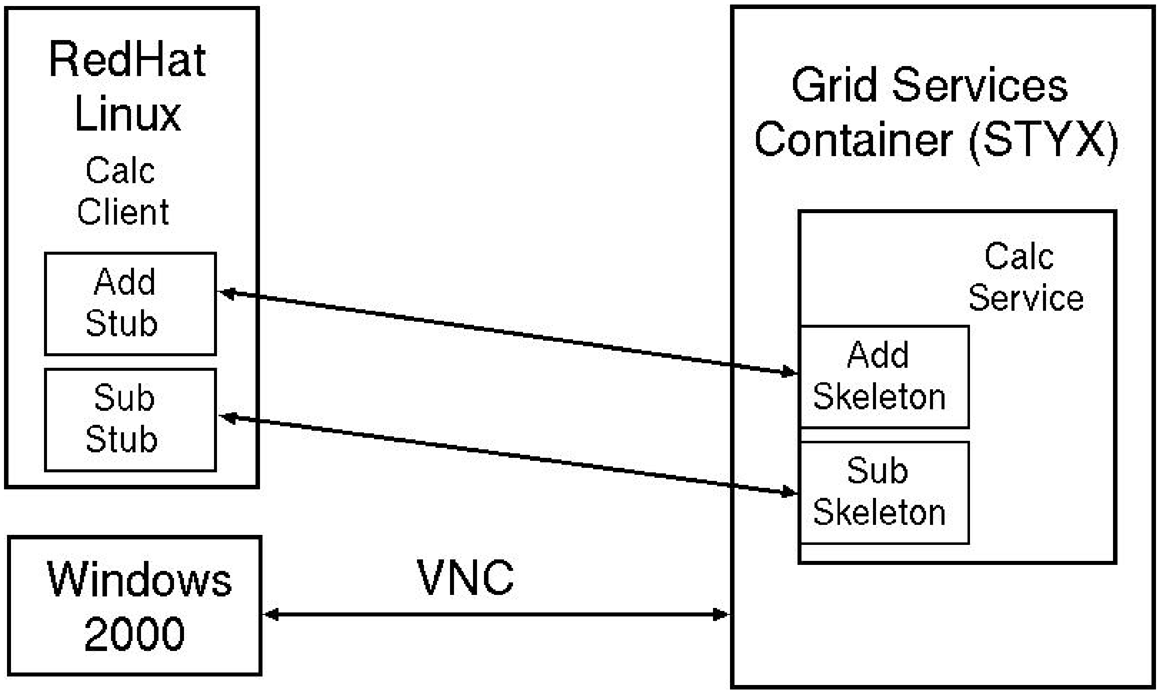


Figure 1: Architecture Diagram

Notice that all references to the counter service are given using global web references that include the name of the system and a path that uniquely identifies a service available through a Grid Services Container. In the example 3 below, The Grid services container is listening on port 8080 on styx and has a service available called ogsa/services/guide/counter/CounterFactoryService/calc which can accept input parameters, passed by the client , to specify which of the available “calc” functions should be invoked.

The following examples show how the grid process works. The first three examples below show how to invoke Grid Services. Examples 4 and 5 demonstrate the built-in capabilities for Grid Management and Administration. Services can be started and stopped using the administrative interfaces to interact with the Grid Services Container. Sample computer outputs from the installation and test are shown below. The four examples demonstrate initializing the grid services environment, creating and accessing a local service, accessing a remote service on styx (from astro), and using the Globus Service Browser to verify services that are active in the Globus Grid Services Container. AFRL/IF also used the Globus notification services to set up a client that is notified whenever the counter changes (not shown in the examples below).

Notice that all references to the counter service are given using global web references that include the name of the system and a path that uniquely identifies a service available through a Grid Services Container. In the example 3 below, the Grid services container is listening on port 8080 on styx and has a service available called ogsa/services/guide/counter/CounterFactoryService/calc which can accept input parameters, passed by the client, to specify which of the available “calc” functions should be invoked.

EXAMPLE 1: Initializing Variables

Example 1 shows how to initialize environment variables that are needed by the grid services container and also by the service browser tool shown in the screengrabs in example 4.

Set up the Grid Services environment

```
cd /usr/local/gt3
. ~scott/.bash_profile        // Setup Java/Grid Environment
. setenv.sh
globus-start-container         // Start Java Container
globus-service-browser        // Start Gui to show the state of services
                              // Screengrab before starting Counter services
                              // and after starting them are shown below.
```

EXAMPLE 2: Creating Counter Service

This example demonstrates creating the counter service and client application access for the counter service.

Create the counter service:

The following two lines contain the command to create the Globus counter service:

```
ogsi-create-service
http://localhost:8080/ogsa/services/guide/counter/CounterFactoryService/calc
```

Use the counter service, adding 10

The following lines contain the command to use the Globus counter service:

```
java org.globus.ogsa.guide.impl.CounterClient
http://localhost:8080/ogsa/services/guide/counter/CounterFactoryService/calc
add 10
>> output >> Counter add: 10
```

EXAMPLE 3: Running Counter Service

We got the Counter service running on styx by doing "ant deployGuide" in the gt3 directory. We added that to the instructions that are given below that show how to run, not deploy, the Counter service.

The lines below show that we were able to use the counter service on styx from a client counter application on astro. Notice that the user command prompt indicates that the client is running on astro but the specified service, provided as an argument to the CounterClient client application, is running on styx.oc.rl.af.mil.

Get the counter value from styx:

The following command gets the counter value from styx:

```
[scott@astro gt3]$ java org.globus.ogsa.guide.impl.CounterClient http://styx.oc
.rl.af.mil:8080/ogsa/services/guide/counter/CounterFactoryService/calc get

Counter value: 10
```

Note that the counter value changes (subtraction performed by a different client program)

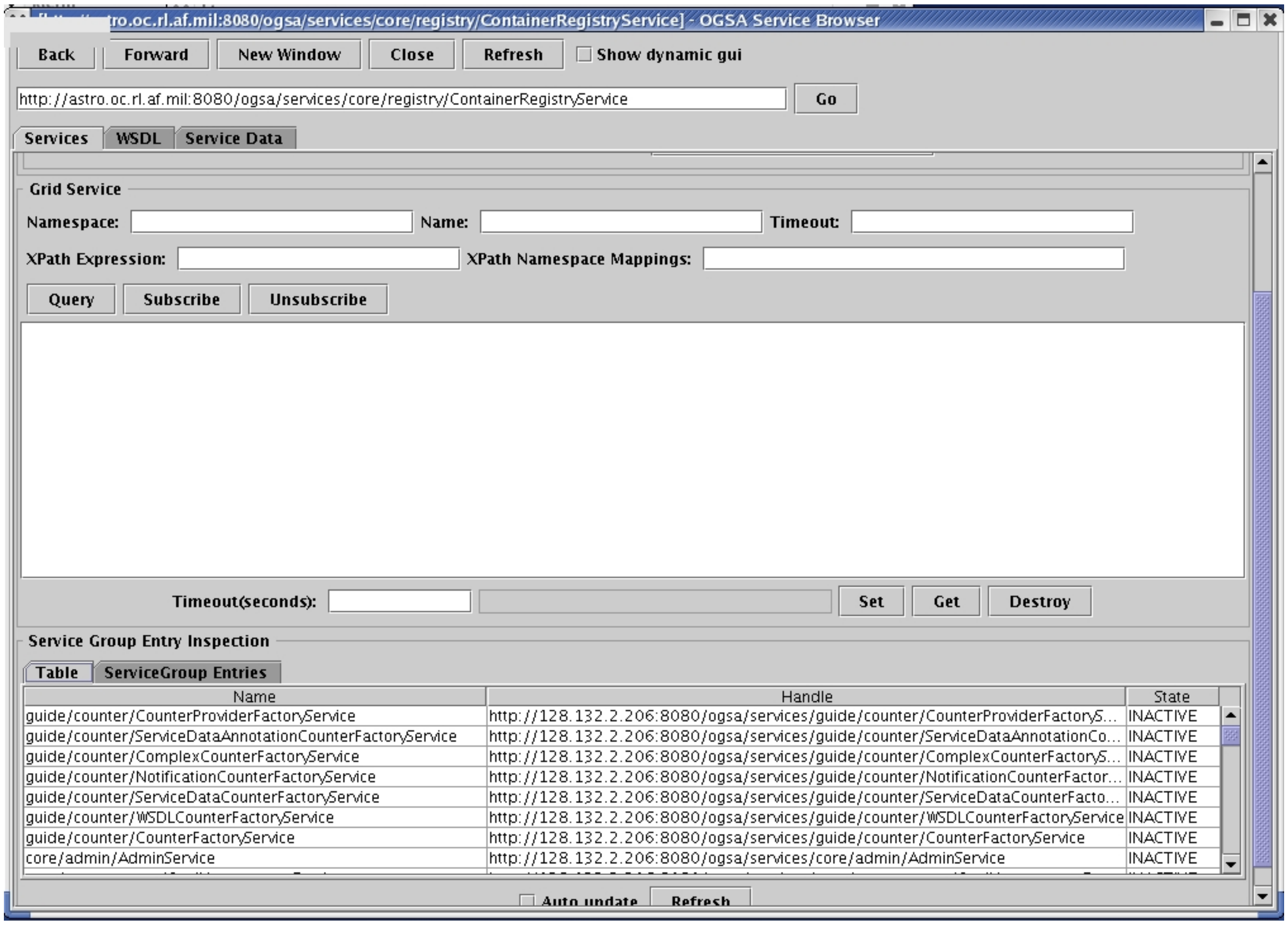
```
[scott@astro gt3]$ java org.globus.ogsa.guide.impl.CounterClient http://styx.oc
.rl.af.mil:8080/ogsa/services/guide/counter/CounterFactoryService/calc get

Counter value: 5

[scott@astro gt3]$
```

EXAMPLE 4: Using Globus Browser

To demonstrate the built-in capabilities for Grid Management and Administration with the Globus Service Browser, Example 4 shows how to use the Globus Service Browser to verify services that are active in the Globus Grid Services Container. Services can be started and stopped using the administrative interfaces to interact with the Grid Services Container. To illustrate use of the browser, Figures 2, 3, and 4 show screenshots of the Globus Services Browser. Each of these figures illustrates an aspect of the functionality available through the Globus Service Browser.

Figure 2 shows that the counter service, including notification services, is inactive (notice CounterFactoryService, WSDLCounterFactoryService, ServiceDataCounterFactoryService, NotificationCounterFactoryService, etc.):

// ServiceBrowserInactiveCounter.jpg shows
// the Inactive counter services



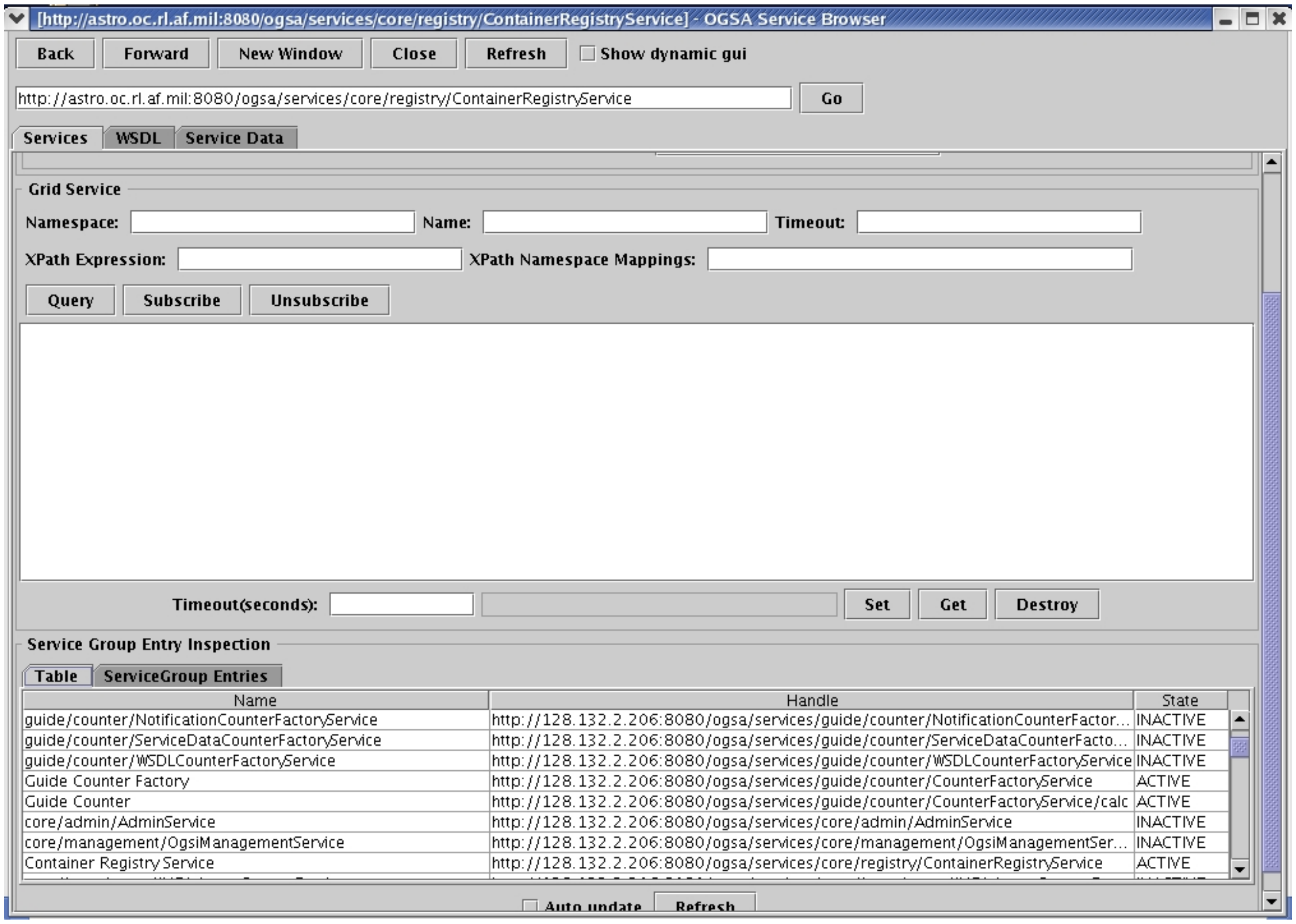
http://astro.oc.rl.af.mil:8080/ogsa/services/core/registry/ContainerRegistryService

Name	Handle	State
guide/counter/CounterProviderFactoryService	http://128.132.2.206:8080/ogsa/services/guide/counter/CounterProviderFactoryS...	INACTIVE
guide/counter/ServiceDataAnnotationCounterFactoryService	http://128.132.2.206:8080/ogsa/services/guide/counter/ServiceDataAnnotationCo...	INACTIVE
guide/counter/ComplexCounterFactoryService	http://128.132.2.206:8080/ogsa/services/guide/counter/ComplexCounterFactoryS...	INACTIVE
guide/counter/NotificationCounterFactoryService	http://128.132.2.206:8080/ogsa/services/guide/counter/NotificationCounterFactor...	INACTIVE
guide/counter/ServiceDataCounterFactoryService	http://128.132.2.206:8080/ogsa/services/guide/counter/ServiceDataCounterFacto...	INACTIVE
guide/counter/WSDLCounterFactoryService	http://128.132.2.206:8080/ogsa/services/guide/counter/WSDLCounterFactoryService	INACTIVE
guide/counter/CounterFactoryService	http://128.132.2.206:8080/ogsa/services/guide/counter/CounterFactoryService	INACTIVE
core/admin/AdminService	http://128.132.2.206:8080/ogsa/services/core/admin/AdminService	INACTIVE

Figure 2: Counter Service Inactive

Figure 3 shows that the counter service, CounterFactoryService, is now active and one instance of the CounterFactoryService “calc” object has been created.

// ServiceBrowserActiveCounter.jpg
// Shows that counter is active
// The CounterFactoryService and one instance
// of the Counter Service are active



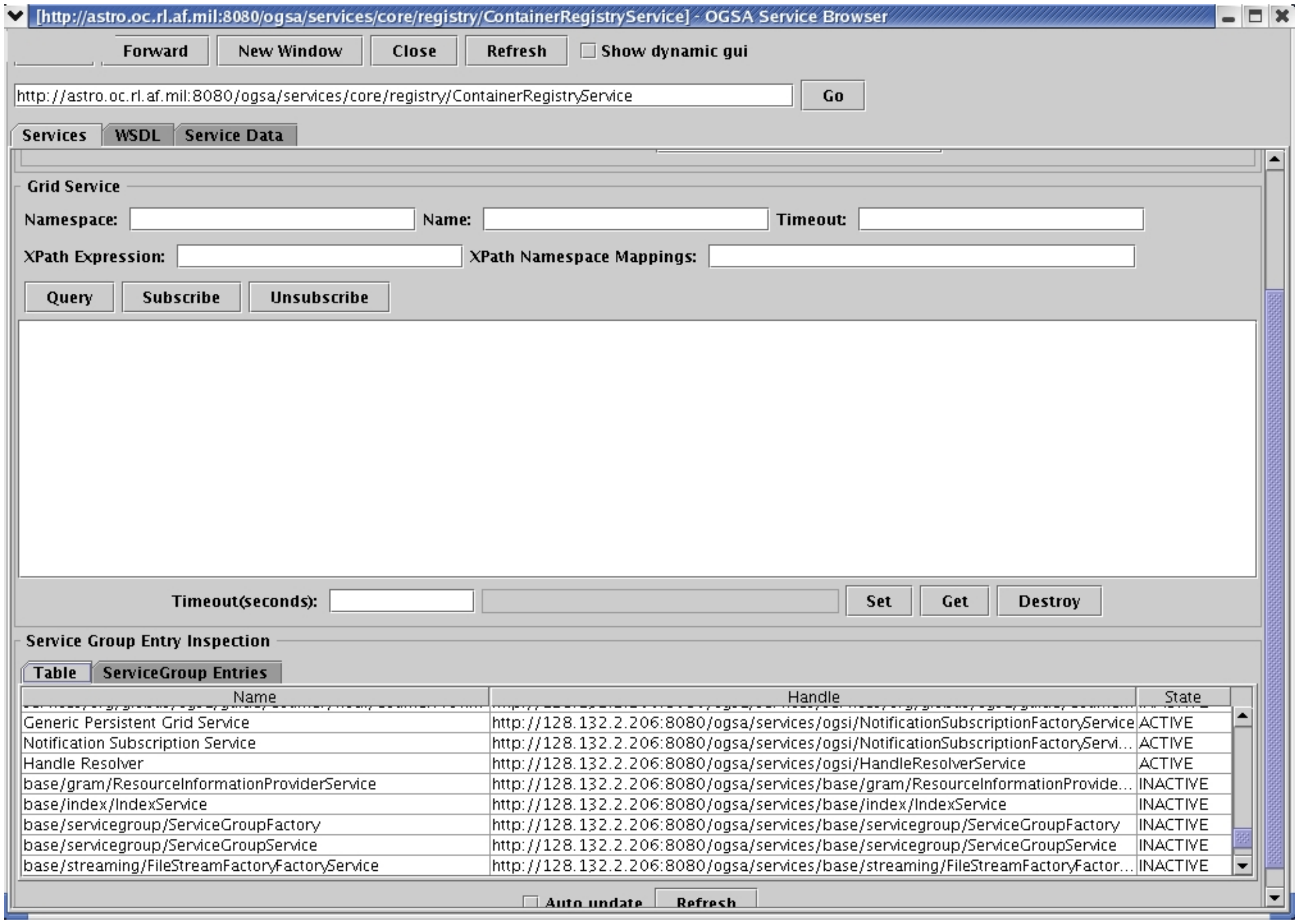
http://astro.oc.rl.af.mil:8080/ogsa/services/core/registry/ContainerRegistryService

Name	Handle	State
guide/counter/NotificationCounterFactoryService	http://128.132.2.206:8080/ogsa/services/guide/counter/NotificationCounterFactor...	INACTIVE
guide/counter/ServiceDataCounterFactoryService	http://128.132.2.206:8080/ogsa/services/guide/counter/ServiceDataCounterFacto...	INACTIVE
guide/counter/WSDLCounterFactoryService	http://128.132.2.206:8080/ogsa/services/guide/counter/WSDLCounterFactoryService	INACTIVE
Guide Counter Factory	http://128.132.2.206:8080/ogsa/services/guide/counter/CounterFactoryService	ACTIVE
Guide Counter	http://128.132.2.206:8080/ogsa/services/guide/counter/CounterFactoryService/calc	ACTIVE
core/admin/AdminService	http://128.132.2.206:8080/ogsa/services/core/admin/AdminService	INACTIVE
core/management/OgsiManagementService	http://128.132.2.206:8080/ogsa/services/core/management/OgsiManagementSer...	INACTIVE
Container Registry Service	http://128.132.2.206:8080/ogsa/services/core/registry/ContainerRegistryService	ACTIVE

Figure 3 Counter Factory Service

Figure 4 shows that the NotificationSubscriptionFactoryServices are active:

// ServiceBrowserOtherServices.jpg
// Shows that Handle Resolver,
// Notification Subscription Service and
// Generic Persistent Grid Service are
// active



[http://astro.oc.rl.af.mil:8080/ogsa/services/core/registry/ContainerRegistryService] - OGSA Service Browser

http://astro.oc.rl.af.mil:8080/ogsa/services/core/registry/ContainerRegistryService

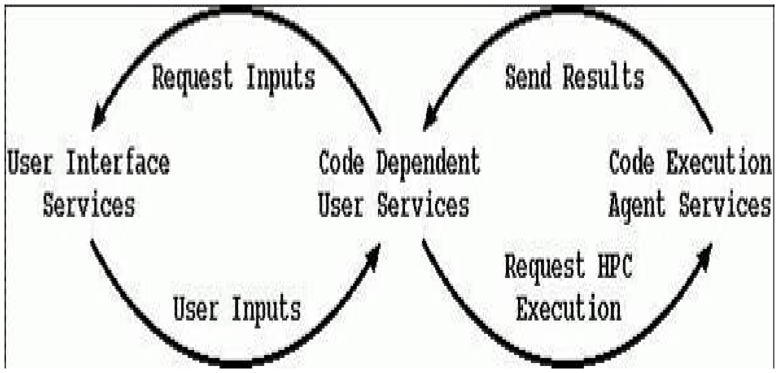
Name	Handle	State
Generic Persistent Grid Service	http://128.132.2.206:8080/ogsa/services/ogsi/NotificationSubscriptionFactoryService	ACTIVE
Notification Subscription Service	http://128.132.2.206:8080/ogsa/services/ogsi/NotificationSubscriptionFactoryServi...	ACTIVE
Handle Resolver	http://128.132.2.206:8080/ogsa/services/ogsi/HandleResolverService	ACTIVE
base/gram/ResourceInformationProviderService	http://128.132.2.206:8080/ogsa/services/base/gram/ResourceInformationProvide...	INACTIVE
base/index/IndexService	http://128.132.2.206:8080/ogsa/services/base/index/IndexService	INACTIVE
base/servicegroup/ServiceGroupFactory	http://128.132.2.206:8080/ogsa/services/base/servicegroup/ServiceGroupFactory	INACTIVE
base/servicegroup/ServiceGroupService	http://128.132.2.206:8080/ogsa/services/base/servicegroup/ServiceGroupService	INACTIVE
base/streaming/FileStreamFactoryFactoryService	http://128.132.2.206:8080/ogsa/services/base/streaming/FileStreamFactoryFactor...	INACTIVE

Figure 4: Notification Subscription Factory Services

These services together constitute the necessary steps to initiate and run an interactive grid job, by initializing the grid services environment, creating and accessing a local service, accessing a remote service on the Grid Services Container, and using the Globus Service Browser to verify services active in the Globus Grid Services Container. This is a significant step accomplishment to move grid computing toward interactive computing..

3. The HIE FrameWork Grid Interface

The HIE FrameWork implements a general purpose Web interface for access to remote applications, usually running on HPCs. The Web interface is driven by objects that implement interfaces to specific codes, allowing the Web interface to collect application-specific inputs and deliver them to remote applications appropriately. This section describes the steps that use grid services to implement FrameWork application interface objects. The application interface object is used by the FrameWork server to determine the variables that have to be supplied through the Web interface. Figure 5 shows the FrameWork flow. In the diagram the Code Dependent User Services represent the code interface object. Notice that the code interface object provides information to the User Interface Services, or FrameWork server, that describes the inputs to be collected from the user. The User Inputs are then used to request execution service as shown in Figure 5.



FrameWork Flow

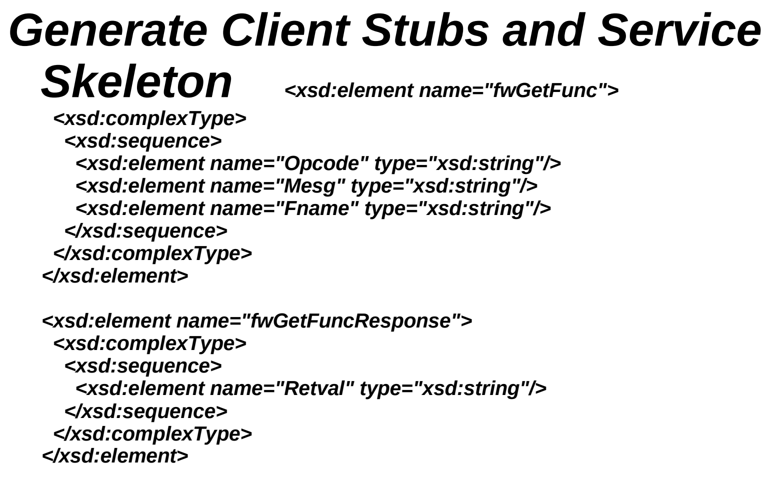
Figure 5: Frame Work Flow

The original FrameWork implementation used a Corba interface to connect the FrameWork server with the code interface object. For this project, we added a Grid interface to allow developers to implement code interface objects using the Globus Grid software or the Corba software. For both Globus Grid and Corba, the code to implement the interface and establish the connection between the FrameWork server and the code interface object can be generated automatically from an interface description that specifies remote functions (implemented in the code interface object) and parameters that are needed to call the functions. The FrameWork server can then access the remote function without regard for network issues, like locating the remote object and opening a connection, by simply calling the function. For the FrameWork, the code interface object is implemented as a service that waits for service requests generated by the Globus Grid or Corba software that is executed as a result of a request by the FrameWork server to execute an interface function.

In the examples below, we focus on the FwGetFunc function that is implemented in the code interface object. After the user selects one of the codes to execute, the code interface object must return a list of functions available for that code. Each of the functions may require a different set of input parameters. The code interface object for the selected code must return the information needed by the FrameWork server so that it can present a list of the available functions to the user, through the Web browser interface. Once a particular function is selected, the FrameWork server uses the same code interface object that also must implement the FwGetParams function, so that the FrameWork server can request the specific list of parameters that are needed by the code from the user, through the Web browser interface.

4. Generating Stubs and Skeletons

For both Globus and Corba, a standard interface description is used to automatically generate the interface functions needed by the client (called a stub) and the server (called a skeleton). Generally, all of the service code needs to be built into the server, by implementing the code that is executed when the service interface (skeleton) is invoked. The term skeleton seems appropriate since the interface code is just a container in which the service must be implemented. The stub, by contrast is just the client’s interface to make the call and pass the parameters to the service implementation. In Corba, the interface is specified using a standard called Interface Definition Language (IDL). The Grid uses an Ex(tensible) M(arkup) L(anguage) XML XSLT specification. Extensible stylesheet language transformation (XSLT) is a language for transforming XML documents into other XML documents. XSLT is designed for use as part of XSL, which is a stylesheet language for XML. The example XML XSLT code for our Grid interface to the fwGetFunc interface for the code interface object is shown in Figure 6.



Generate Client Stubs and Service Skeleton

```
<xsd:element name="fwGetFunc">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Opcode" type="xsd:string"/>
      <xsd:element name="Mesg" type="xsd:string"/>
      <xsd:element name="Fname" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="fwGetFuncResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Retval" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure 6: Specifying the Interface

5. A Globus Function Stub

The stub is used by the client to call a function that is actually implemented as a service in the grid services container. The code shown in Figure 7 was generated by a Globus utility program for the fwGetFunc interface to the code interface object that is described above.

```
  fwGetFunc.Opcode = argv[2];
  fwGetFunc.Mesg = argv[3];
  fwGetFunc.Fname = argv[4];

   /* create blog resource with createBlogTopic
operation */
   result = Fw_fwGetFunc(
       fw_handle,
       argv[1],
       &fwGetFunc,
       &fwGetFuncResponse,
       &create_fault_type,
       &fault);



 /* destroy response from fwGetFunc */
   fwGetFuncResponseType_destroy(fwGetFuncResponse);

   printf("Returning from fwGetFunc\n");

   /* destroy client handle */
   FwService_client_destroy(fw_handle);

   rc = globus_module_deactivate(FWSERVICE_MODULE);
   if(rc != 0)
   {
      globus_fatal("FwService deactivate failed");
   }

   exit(0);
}
```

Figure 7: Globus Client Stub

6. A Globus Function Skeleton

The skeleton function is compiled and installed in the C Web Services Container. Users can add backend code to perform desired services. The part of the skeleton shown in Figure 8 initializes debugging and can be used to perform custom initialization of the service if necessary. The Fw_fwGetFunc_imp function implements the service that is compiled into the C Web Services Container and sends results back to the client program, through the client stub interface, by copying it into an "xsd_string Retval" variable generated from the interface description.

```
FwService_init                   FwService_finalize
Fw_SetTerminationTime_init       Fw_SetTerminationTime_impl
Fw_Destroy_init                  Fw_Destroy_impl
Fw_GetCurrentMessage_init        Fw_GetCurrentMessage_impl
Fw_Subscribe_init                Fw_Subscribe_impl
Fw_fwGetResults_init             Fw_fwGetResults_impl
Fw_fwExecFunc_init               Fw_fwExecFunc_impl
Fw_fwGetParams_init              Fw_fwGetParams_impl
```

```
Fw_fwGetFunc_init(
    globus_service_engine_t             engine,
    globus_soap_message_handle_t        message,
    fwGetFuncType * fwGetFunc)
{
    /* add function local variable declarations here */
    globus_result_t                     result = GLOBUS_SUCCESS;

    /* initialize trace debugging info */
    GlobusFuncName(Fw_fwGetFunc_init);
    FwServiceDebugEnter();

    /*
     * If no configuration or initialization needs to be done, this
     * call can remain empty.
     */

    FwServiceDebugExit();
    return GLOBUS_SUCCESS;
}
globus_result_t
Fw_fwGetFunc_impl(
    globus_service_engine_t             engine,
    globus_soap_message_handle_t        message,
    globus_service_descriptor_t *       descriptor,
    fwGetFuncType * fwGetFunc,
    fwGetFuncResponseType * fwGetFuncResponse,
    const char ** fault_name,
    void ** fault)
{
    /* add function local variable declarations here */
    globus_result_t                     result = GLOBUS_SUCCESS;


/* initialize trace debugging info */
   GlobusFuncName(Fw_fwGetFunc_impl);
   FwServiceDebugEnter();

   /* This is where it all happens.  Service implementer must
    * implmenent this function.  Asume that fwGetFunc has
    * been initialized and filled with request values.
    * fwGetFuncResponse must be set by the implementer.
    */

   /* Use the error object construction api */
   result = FwServiceErrorNotImplemented("Fw_fwGetFunc");

   FwServiceDebugExit();
   return result;
}
```

Figure 8: Globus Service Skeleton

7. SUMMARY:

This research project successfully set up and operated an interactive grid using Globus Toolkit, ran it, and prepared the way for subsequent integration with the HIE framework. This is a significant step up from traditional HPC grid architectures, which are designed for batch applications, where the user lacks direct control over the execution of their application. Applications, such as the HIE framework require a near real-time response and an interactive environment. Lessons learned were how to successfully use the Globus toolkit to run these services, including initializing variables, creating and running the counter service, and using the Globus browser. Potential future work in the area would be to use this capability to run interactive grid based work, supporting the warfighter in areas such as the Joint Battlespace Infosphere.

8. REFERENCES:

1**.** Ramseyer, G.O., Linderman, R.W., Spetka, S.E., Fitzgerald, D.J. , Moore, M.J., "Rapid Remote Hyperspectral Image Exploitation on High-Performance Computers", Wailea Marriott, Wailea, Maui, Hawaii, September 8-13, 2003.

2. Ramseyer, G.O., Linderman, R.W., and Spetka, S.E., “Open Architecture for Large Imaging Systems”, C4ISR Architectures, Las Vegas, NV Oct 18-19, 2004.

3. Ramseyer, G.O., Phister, P.W., Spetka, S.E., Linderman, R.W., "Rapid C4I High-Performance Computing for the Joint Battlespace Infosphere", Airborne C4I Conference, London, England, October 2002.

4. Ramseyer, G.O., Spetka, S.E., Linderman,R.W., Fitzgerald, D.J. , Moore, M.J., "Open-Architecture Middleware for Hyperspectral Image Exploitation", 28th Annual GOMACTech (Government Microcircuit Applications & Critical Technology) Conference, "Countering Asymmetric Threats", Hyatt Regency Tampa, Tampa, FL, March 31-April 3, 2003.

5. Ramseyer, G.O., Spetka, S.E., Linderman, R.W., Romano, B.C., "The FrameWork: An Open-Architecture for Very Large Image Exploitation", 2002 Command and Control Research and Technology Symposium, Naval Postgraduate School, Monterey, CA, June 11-13, 2002.

6. Ramseyer, G.O., Spetka, S.E., Linderman, R.W., Romano, B.C., "Rapid C4I High Performance Computing for Hyperspectral Imaging Exploitation", 6th International Command and Control Research and Technology Symposium, U.S. Naval Academy, Annapolis, MD, June 19-21, 2001.

7. Ramseyer, G.O., Spetka, S.E., Linderman, R.W., Fitzgerald, D.J., "Integrated High-Performance Computing of Hyperspectral Imaging Algorithms", DOD High-Performance Computing Modernization Program Users Group Conference 2004, Williamsburg, VA, June 7-11, 2004.

8. Spetka, S. E., Ramseyer, G. O., Linderman, R. W., “Using Globus Grid Objects to Extend a Corba-based Object-Oriented System”, OOPSLA 05, San Diego, CA, October 16-20, 2005.

9. Spetka, S.E., Ramseyer, G.O., Linderman, R.W., "Grid Technology and Information Management for Command and Control", 10th International Command and Control Research and Technology Symposium, The Future of C2, McLean, Virginia, VA, June 13-16, 2005.

10. Spetka, S.E., Ramseyer, G.O., Linderman, R.W., "Redeveloping a High-Performance Computing FrameWork", 18th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), ACM Special Interest Group on Programming Languages, Anaheim Convention Center, Anaheim, California, October 26-30, 2003.

11. Spetka, S.E., Ramseyer, G.O., Linderman, R.W., "A FrameWork for High-Performance Image Exploitation", 17th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), ACM Special Interest Group on Programming Languages, Washington State Convention & Trade Center, Seattle, Washington, November 4-8, 2002.

12. Spetka, S.E., Ramseyer, G.O., Linderman, R.W., Moore, M.J., "A Software FrameWork for HPEC System Development", Sixth Annual Workshop on High Performance Embedded Computing, MIT Lincoln Laboratory, September 24-26, 2002

13. Spetka, S.E., Ramseyer, G.O., Fitzgerald, D.J., Linderman, R.W., "A Distributed Parallel Processing System for Command and Control Imagery", 7th International Command and Control Research and Technology Symposium, Quebec City, QC, Canada, September 16-20, 2002.

14. “Web Services Architecture, W3C Working Draft 8 August 2003”, http://www.w3.org/TR/ws-arch/ .